OPTIMIZING QUERIES USING A MATERIALIZED VIEW IN A DATA WAREHOUE

By

JING HU

Bachelor of Medicine

Anhui Medical University

Anhui, P. R. China

1998

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2006

OPTIMIZING QUERIES USING A MATERIALIZED

VIEW IN A DATA WAREHOUE

Thesis Approved:

Dr. G. E. Hedrick

Dr. K. M. George

Dr. N. Park

A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation and acknowledgement to my advisor Dr. G. E. Hedrick for his supervision, assistance, guidance and patience in completing my thesis work. I genuinely thank him for everything I achieved in my research so far. This thesis could not have been completed without his constructive comments that significantly improved the presentation and contents of my thesis.

My sincere appreciation goes to my committee members, Dr. N. Park and Dr. K. M. George for their supervision and guidance.

Special thanks go to my parents in China for their enthusiasm to educate their children and for their love and support.

I would also like to extend my acknowledgement to special thanks go my friends: Yue Wang, Zhu Wang, for their enthusiasm and help in difficult times.

Finally, and most importantly, I would like to thank my brother, Yong Hu, sister-in-law, Jing Ding for their support, encouragement and love in the duration of my studies in OSU.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figures Page

CHAPTER I

INTRODUCTION

A data warehouse is the data depositary for a collection of external data sources and internal data sources to support data analysis. It supports knowledge workers (managers or analysts) in making strategic decisions better and faster than they could otherwise. In order to provide managers or analysts information quickly and efficiently, the performance of using the data warehouse needs to be as efficient as possible. Many techniques of increasing performance have been proposed or implemented recently. They relate to variety of data management problems that include query optimization, maintenance of physical data independence, data integration, and data warehouse design [H00]. This thesis surveys the problem of optimizing queries by query rewriting by using a materialized view.

A materialized view[1] (MV) is similar to a view but the data is actually stored on disk (view that materializes). A materialized view provides access to table data by storing the results of a query in a separate schema object. Unlike an ordinary view that does not take up any storage space or contain any data, a materialized view contains the rows resulting from a query against one or more base tables or views. A materialized view can be stored in the same database as its base tables or in a different database. Materialized views are often used for summary and pre-joined tables, or just to make a snapshot of a

[1] www.orafaq.com/glossary/faqglosm.htm October, 2005

table available on a remote system. A MV must be refreshed when the data in the underlying tables is changed. The materialized view has recently received significant attention [H01].

Query rewriting using materialized views considers the problem of replacing an original query with a new expression containing the materialized views such that the new query is equivalent to the original one. Query rewrites are particularly useful in a data warehouse environment. Informally, a view can be useful for a query if the set of relations it mentions overlaps with that of the query, and it selects some of the attributes selected by the query, and the constrains of the view must be equivalent or weaker than those from the query. The example is the following. Suppose we are given a query Q over a database schema, and a set of views $V_{1,}\ldots, V_m$ over the same schema. Is it possible to answer the query Q using only the answers to the views $V_{1,}\ldots, V_m$, and if so, how?

This thesis focuses on the problem of answering queries using views for select-project-join queries under set semantics. While such queries are quite common in data integration applications, many applications need to deal with queries involving grouping and aggregation, semi-structured data, nested structures and integrity constraints. Our focus is on an algorithm for answering queries using views. Hence, we begin with the class of select-project-join queries.

We consider the problem of answering conjunctive queries using a set of conjunctive views ((i.e., select-project-join queries) in the presence of a large number of views. Several algorithms have been used to solve this problem. One of the most popular algorithms is the bucket algorithm that was used in the Information Manifold system

[LRO96]. Other notable algorithms include the inverse-rules algorithm [DG97] and the System-R style optimizer [CKPS95]. We briefly state these algorithms in the related work.

Based on the insights into the previous algorithm, this thesis presents the containment bucket algorithm that requires only the availability of containment information among the views in order to replace the relations in a user query. The algorithm proposed in this thesis addresses the following issue: (1) Finding view rewritings that are equivalent to the original view. (2) Using semantic containment information for replacing the relations in the original query. The key idea underlying the containment bucket algorithm is a change of perspective: instead of building rewritings by combining rewritings for each query subgoal or database relation, we consider the containment information in the query can interact with the available views. The result is that the second phase of the containment bucket algorithm needs to consider drastically fewer combinations of views.

The rest of this thesis is organized as follows: Section 2 introduces the related work. Section 3 discusses the preliminaries. Section 4 presents the bucket algorithm. Section 5 contains a brief idea of containment bucket algorithm. Section 6 describes the experimental results and Section 7 discusses the conclusion and future works.

CHAPTER II

RELATED WORK

The problem of finding query writing algorithms has been used in data warehousing to support to the speed up of query evaluation. In order to obtain performance benefits from query writing, choosing an appropriate set of views to materialize in the database is crucial. Other approaches include using the inverse-rules algorithm or a System-R style optimizer. The inverse-rules algorithm [DG97] constructs a set of rules that invert the view definitions. The rules show how to compute tuples for the database relations from tuples of the views. However, it might invert some of the useful computations done to produce a view so the second stage is almost as expensive as the bucket algorithm's exponential conjunctive-query-containment test. The System-R style optimizer [CKPS95] is a rule-based query optimizer which uses a map table that specifies which subexpressions of the query can be substituted by one of the available views. The map table is computed before join enumeration begins. During a join enumeration, rewrites of the expression under consideration are located in the map table and their cost estimated in the normal way.

CHAPTER III

PRELIMINARIES

3.1 Data warehouse concepts and architecture

A data warehouse is a “subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management’s decision making process” [CBC05] which contains historical, summarized and consolidated data. Date warehouse are aimed at decision support and constitute the background to enable business intelligence to let companies access, analyze and share information with employees internally or with customers, suppliers and partners externally.

The structure of a data warehouse is shown in Figure 1.

The back end includes mapping the data source into the data warehouse and ensures building the warehouse database. The data source which needs to be fed to data warehouse can be extracted from database systems of each department of the company or from external source such as the information from other companies, results of surveys or internet etc [JLVV00]. Data warehousing system use the back end tools to extract, clean, load and refresh data for populating warehouse. Query-based Tools for data integration is introduced in [RBN02].

A data mart is subset of a data warehouse that is normally in the form of summary data and focuses on a department. Each department has its own interpretation of what a data mart should look like and each department's data mart is specific to its own needs.

The data marts from different departments which can exist by themselves or integrated together to the enterprise-wide data warehouse are usually easier to implement than a data warehouse. The data mart contains only modest amounts of historical information and provides the needs of the department. It is typically housed in multidimensional technology which is great for flexibility of analysis but is not suitable for large amounts of data. Users can get faster response times from data mart than from data warehouse.

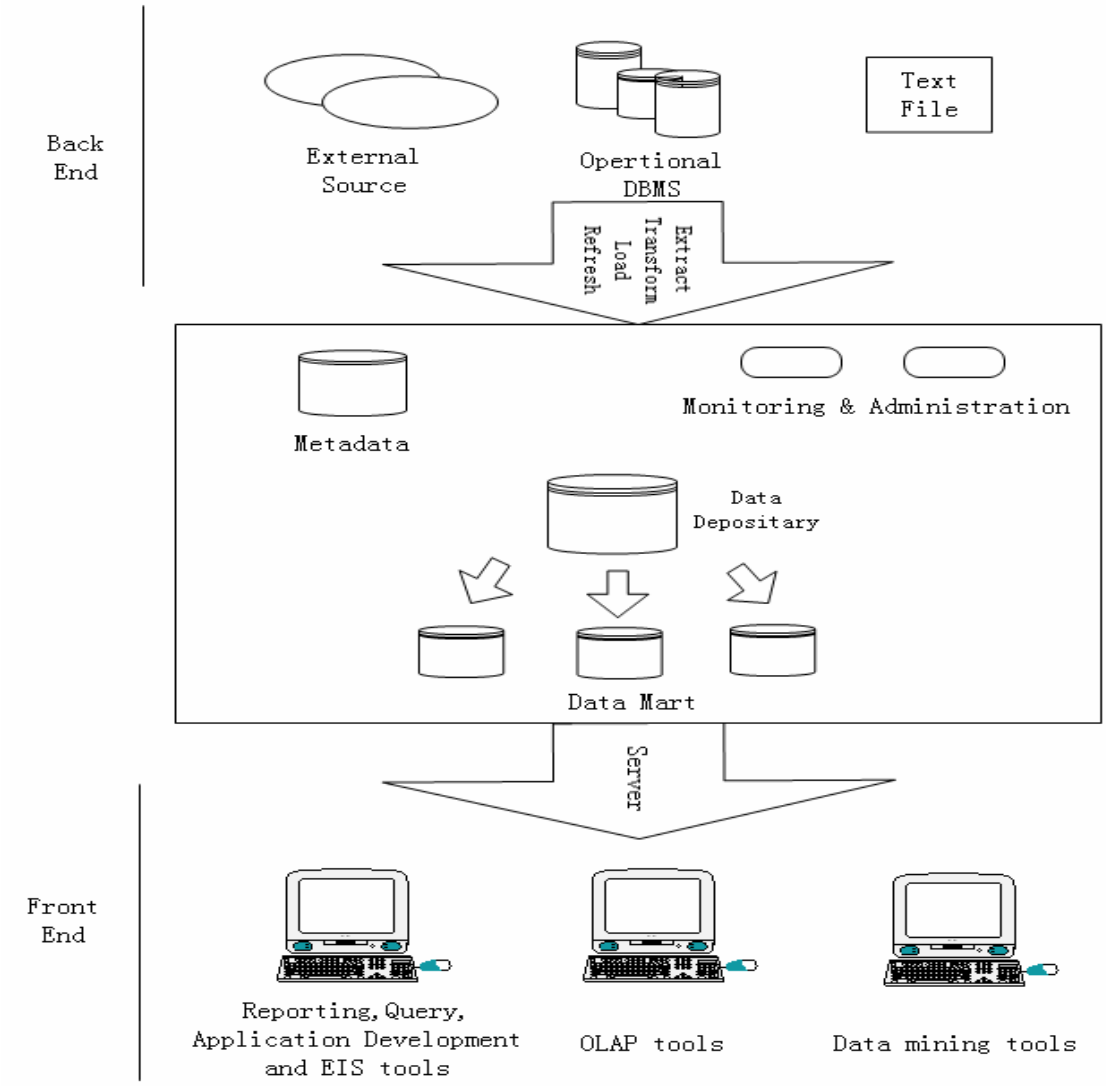

Figure 1. Data Warehouse Architectures

At this point, an enterprise-wide data warehouse either can be a virtual data warehouse that has a common interface to data warehouse, but the queries are delegated to the data marts, or a materialized data warehouse which has loaded the data from the data marts as well as other information sources. On-Line Transaction Processing (OLTP) [CBC05] system is usually separated from the data warehouse which is important for the daily business of the enterprise. Therefore, the data warehouse must be adapted to any changes which occur in the underlying data sources, such as changes of the schema, the physical location of a data source or a change of the time window for the extraction of source data.

Metadata is data about data which is used in data acquisition/collection, data transformation, and data access. The metadata should include: *administrative metadata* which includes all information for building and using data warehouse, *business metadata* which includes the ownership of data, business terms and definitions, *operational metadata* which includes procedures on how a data warehouse is used and accessed, the data flow in the data warehouse, authorizations on who is responsible for and who has access to the data in the data warehouse and data in the operational system, and monitoring the usage statistics error reports and data transformation [CD97].

The front end provides the restitution of data from data warehouse to fulfill user demands via query tools. Data warehouse supports information analysis, often known as online analytical processing (OLAP) which provides advanced analysis tools to extract information from data stored in data warehouse. OLAP is design to provide aggregate information that can be used to analyze the contents of databases and data warehouse. The managers and analysts request the information frequently to support them in the

daily business work of an enterprise, so they need fast response time for their queries and high quality data. Beyond the OLAP operations, other possible client applications on the front end include [JLVV00] [CBC05]:

- Reporting and query tools.
- Application development tools.
- Geographic information systems (GIS).
- Data mining tools.
- Decision support systems (DSS).
- Executive information systems (EIS) tools.

In this thesis the problem considered is how to improve the query response time in the front end of data warehouse. We deal with optimizing queries using materialized views to improve query performance.

3.2 Performance of data warehouse

To meet user demands for more timely and flexible analysis, the performance of a data warehouse is an important problem and several related issues arise.

Morzy, T.and Wrembel, R. proposes a *multiversion data warehouse* which can handle dynamic changes in their structure and content in [MW04]. In [SB00], Saharia, A. N. and Babad, Y. M. propose an enhancement data warehouse structure, by building additional intelligence in the form of an adaptive and efficient query *cache*. Recently, the *index data structure* is mostly used for OLAP and data warehouse application [LR99] [OL01] [SWS02].

Using a materialized view (MV) to accelerate OLAP queries is one of the most common methods used in data warehousing which is computing the answer to a query based on a set of materialized view, instead of on the raw data in the database. A query rewrite is a mechanism where applications from the end user or database transparently to improve query response time by rewriting the SQL query using the materialized view instead of accessing the original tables.

The thesis focuses on optimizing queries using materialized views for queries in the data warehouse. We deal with queries rewriting. Indeed, the problem of answering queries using views has been considered in these contexts as well [LMSS95] [AD98] [AGK99]. We only deal with the class of join-select-project queries, and consider the equivalent rewriting from views.

3.3 Materialized view

Materialized views [G03] are similar to table objects, and not merely a simple query, they have certain benefits over simple views [D00] and normally are used in situations where performance is a main concern. Materialized views which consume storage space stored in the same database as their base tables can improve query performance through query rewrites. In warehousing applications, large amounts of data are processed and similar queries are frequently repeated. If these queries are pre-computed and the results stored in the data warehouse as a materialized view, then using materialized views significantly improves performance by providing fast lookups into the set of results. A materialized view is used to eliminate overhead associated with expensive joins or aggregations for a large number of queries. The purpose of a materialized view is to

increase query execution performance. The existence of a materialized view is transparent to SQL applications, so that a DBA can create or drop materialized views at any time without affecting the validity of SQL applications. The types of materialized views are materialized views with aggregates, materialized views containing only joins and nested materialized views. In this thesis, we only consider materialized views containing only joins. The advantage of creating this type of materialized view is that expensive joins will be precalculated.

3.4 OLAP

On-Line Analytical Processing (OLAP) provides advanced analysis tools to extract information from data stored in data warehouse. A data warehouse together with tools such OLAP and/or data mining are collectively referred to as business intelligence technologies. OLAP is a term that describes a technology that uses a multi-dimensional view of aggregate data to provide quick access to strategic information for the purpose of advanced analysis. OLAP enables users to gain a deeper understanding and knowledge about various aspects of their corporate data through fast, consistent, interactive access to a wide variety of possible views of the data and enables to help knowledge workers (managers or analysts) to make future action. A typical OLAP calculation can be more complex than simply aggregating data. OLAP is design to provide aggregate information that can be used to analyze the contents of databases and data warehouse. An increasingly popular data model for OLAP applications is the multidimensional database, also known as data cube. Multidimensional models lend themselves readily hierarchical views in roll-up display and drill-down display [VS99].

3.5 Query writing using view

In this thesis, we consider that the relations used in the queries are database relations which deal with OLAP tools in the data warehouse, and the problem of answering queries using views for conjunctive queries (i.e., select-project-join queries) [U88]. A conjunctive query has the form:

$q(\overline{X}) :- r_1(\overline{X}_1), \ldots, r_n(\overline{X}_n)$

Where q is query, $r_1, \ldots, r_n$ are base table names and $\overline{X}, \overline{X}_1, \ldots, \overline{X}_n$ are either variables or constants. The head of the rule is $q(\overline{X})$ and refers to the answer relation. The atoms $r_1(\overline{X}_1), \ldots, r_n(\overline{X}_n)$ are subgoals in the body of the query, where $r_1, \ldots, r_n$ are database relations. We require that all rules be *safe* which means every variable that occurs in the head must also occur in the body, i.e. $\overline{X} \subseteq \overline{X}_1 \cup \ldots \cup \overline{X}_n$.

We use $V, V_1, \ldots, V_m$ to denote views that are defined on the database relations. The relationship between a query Q and its rewriting R is not simply equivalence of queries because the views are not additional database relations, but they are defined by conjunctive queries. We give some definitions below.

3.6 Definitions [H01]

Definition 1. Containment and equivalence: A query Q_1 is said to be contained in a query Q_2, denoted by $Q_1 \subseteq Q_2$, if for all databases D, the set of tuples computed for Q_1 is a subset of those computed for Q_2; i.e., $Q_1(D) \subseteq Q_2(D)$. The two queries are said to be equivalent if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$.

Definition 2. Equivalent rewritings: Let Q be a user query and V = {V1, … , V_m} be a set of view definitions. The query Q' is an equivalent rewriting of Q using V if:

- Q' refers only to the views in V,
- Q' is equivalent to Q (produce the same answer of any given database).

The following example shows:

Q: q(x, u) :- r(x, y), r_0(y, z), r_1(x, w), r_2(w, u).[2]

V: V_1(a, b) :- r(a, c), r_0(c, b), r_1(a, d)

V_2(a, b) :- r_1(a, c), r_2(c, b), r_0(d, e)

Then, the equivalent query using V is shown as follows:

Q' : q(x, u) :- V_1(x, z), V_2(x, u).

Definition 3. Distinguished variable/non-distinguished variable: A distinguished variable is a variable that occurs in the head of a rule and a non-distinguished variable is a variable that occurs in the body of a rule but not in the head.

In the query: Q: q(x, u) :- r(x, y), r_0(y, z), r_1(x, w), r_2(w, u), x and u are distinguished variables, y, z, w are non-distinguished variables.

Definition 4. Containment-target view [CL03]: A conjunctive view V is a containment-target view if V covers at least one subgoal of query Q. It means the

[2] We only use select-project-join query without restriction on the variables to illustrate these definitions.

distinguished variables of the view V are same as the variables of at least one subgoal *g* of query Q, we say V covers the query subgoal *g*, then we find all views $V_1, \ldots, V$ which can cover every subgoal, these views $V_1, \ldots, V$ are containment-target views for the query Q. We illustrate it as following example:

Q: q (m, c) :- r(d, m), s(d, c)[3]

V: V_1 (d, m) :- r(d, m).

V_2 (d, c) :-r(d, m), s(d, c)

V_3 (d) :- r(d, m), s(d, c)

The view V_1(d, m) covers the query subgoal r(d, m), view V_2(d, c) covers the query subgoal s(d, c). Thus V_1 and V_2 are containment-target views for the query.

[3] We only use select-project-join query without restriction on the variables to illustrate these definitions.

CHAPTER IV

BUCKET ALGORITHM

The goal of all query rewriting algorithms is to rewrite the user query into a query which refers directly to the view sources. However, the number of possible rewritings of the user query using the views is exponential in the size of the query. The Bucket algorithm attempts to rewrite a query using views instead of the logical schema of the predicates. The key idea of the bucket algorithm is that the number of query rewritings that must be considered can be reduced drastically by considering each subgoal of the query in isolation to determine which views may be relevant to a particular subgoal.

The bucket algorithm proceeds in two steps. In the first step, it creates a bucket for each subgoal *g* to contain the views that are relevant for that particular sub-goal. The view is put in the bucket of the subgoal g if it contains this relation and the constraints in the view are compatible or weaker than those in the user query. If a subgoal *g* unifies with more than one subgoal in a view V, then the bucket of *g* will contain multiple occurrences of V. In the second step, the algorithm selects one view from each bucket and combines them into a conjunctive rewriting by performing a Cartesian product of each view of the buckets. All possible candidate solutions are generated, then the algorithm checks whether it is contained in the user query.

We illustrate the bucket algorithm with the following query and views:

Query: Q(c):- Course (c), Student (s), Advised (s, “Dr. Smith”), Registered (s, c)

V: V_1 (s):- Course(c), Registered (s, c)

V_2 (s, c):- Course (c), Registered (s, c)

V_3 (c):- Student (s), Advised (s, “Dr. Smith”), Registered (s, c)

V_4 (s, t):- Student (s), Advised (s, t)

V_5 (d):- Dept (s, d), Student (s), Advised (s, “Dr. Smith”)

The base tables are course, student, advised, registered. Queries and views are defined as conjunctives of these tables. The query asks for all courses taken by students whose advisor is Dr. Smith. We want to rewrite the query using the available views. There are five views of various information sources.

In the first step, the bucket algorithm creates a bucket for each subgoal in Q(c), The Sub-goals in Q(c) are: Course (c), student (s), Advised(s, “Dr. Smith”), Registered (s, x). The bucket for each subgoal g contains the views that include subgoals to which g can be mapped in a rewriting of the query. For example, both the view V_1(s) and V_2 (d, c) contain subgoal Course(c), so the bucket for Course(c) has V_1(s) and V_2 (d, c).

Table 1 lists all the buckets generated by the bucket algorithm.

Course(c)	Student (s)	Advised(s, “Dr. Smith”)	Registered (s, c)
$V_1(s)$	$V_3(c)$	$V_3(c)$	$V_1(s)$
$V_2(s, c)$	$V_4(s, t)$	$V_4(s, t)$	$V_2(s, c)$
	$V_5(d)$	$V_5(d)$	$V_3(c)$

Table 1. The buckets of bucket algorithm

In the second step, the algorithm considers conjunctive query rewritings, each consisting of one conjunctive form every bucket. Specifically, for each element of the Cartesian product of the buckets, the algorithm constructs a conjunctive rewriting and checks whether it is contained or can be made to be contained by adding join predicated in the query. If so, the rewriting is added to the answer. Here, the algorithm generates the candidate solution by performing a Cartesian product of the views in the buckets, the number of candidate solutions is (2 x 3 x 3 x 3) = 54. Each solution is checked by the bucket algorithm if it equals the user query and it selects a correct solution. Hence, the rewriting query contains a union of conjunctive views.

The main disadvantage of bucket algorithm is that there is no interaction between view subgoals, and the algorithm considers each subgoal is isolated. As the result, the buckets contain irrelevant views. Another problem is that the number of candidate solutions that the algorithm generates by executing a Cartesian product of the views in the bucket is too large. Thus, the second step becomes very expensive. In the next section, we show we can reduce the number of candidate solutions to improve the query writing performance.

CHAPTER V

CONTAINMENT BUCKET ALGORITHM

Like the bucket algorithm, the task of query rewriting is accomplished in two steps:

- Bucket Construction
- Solution Generation

The key idea underlying the containment bucket algorithm is to use containment target views to fill each bucket and avoid a containment check. In the Bucket Construction stage, bucket construction, each bucket contains only views that cover the subgoal in which containment target view variables appear. In other words, the variables in subgoal should be distinguished variable in the view(s). In the Solution Generation stage, one view is selected from each bucket in the set of buckets and the conjunction of the selected views is used to generate a solution. All possible combinations of views selected from each bucket are used to generate all solutions related to one set of buckets.

We use same example which used in bucket algorithm to illustrate containment algorithm. The algorithm starts like bucket algorithm. In the first step, the algorithm also needs to create a bucket for each subgoal in Q(c), and puts a containment-target view(s) in the buckets. The Sub-goals in Q(c) are: Course (c), student (s), Advised(s, “Dr. Smith”), Registered (s, x). The bucket for each subgoal g contains the view that includes the containment-target view only. For example, both the view V_1(s) and V_2 (s, c) contain

subgoal Course(c) in bucket algorithm, however, from the definition of containment-target view, only V_2 (s, c) is the containment-target view for subgoal Course(c), so the bucket for Course(c) only has V_2 (s, c). As seen in the example, we must consider mappings from the query to specializations of the view, where the head variables in the subgoal must be in the head of the views.

All the buckets generated by the containment bucket algorithms are shown in table 2.

Course(c)	Student (s)	Advised(s, "Dr. Smith")	Registered (s, c)
V_2(s, c)	V_4(s, t)	V_4(s, t)	V_2(s, c)

Table 2. The buckets using the containment bucket algorithm

In the second step, the algorithm only combines the view of in each bucket, removes the duplicated views, and generates a conjunctive rewriting by combining the views. A set of buckets is chosen such that each subgoal is represented by one and only one bucket in the set. From each bucket a view is selected. In our example, we combine all views: V_2(s, c), V_4(s, t), V_4(s, t), V_2(s, c), and remove the redundant views, then we can get V_2(s, c), and V_4(s, t), and combine it to get rewritten query for user query. Consequently, the solution to the query is expressed as a conjunctive query which consists of the conjunct of the selected views. The solution generated by the process mentioned above is a sound solution and there is no need for performing a containment check. In comparison to the bucket algorithm that generated 54 candidate solutions, we generate only one valid solution. Q(c):-- V_2(s, c), V_4(s, t).

The reason for our algorithm is that it enables us to focus in the second phase only on rewritings where the buckets do not cover the overlapping views. This yields more efficient in the second phase. Our algorithm considerably speeds up the query rewrite stage of an information integration system. This enables the construction of scalable integration systems that can handle large amounts of information.

CHAPTER VI

EXPERIMENTATION RESULT

All experiments were run on a computer with Intel Pentium IV processor 2.2 GHz, 256MB RAM, SQL Sever 2000 database management system, running on Windows XP. All of the algorithms were implemented in Java and compiled to executable.

The test database for our simulation is from US Department of Labor[4]. We create 5 base tables which are *Employer, Job, Applicant table, CaseInfo* and *Wage* tables. The largest table has 65,536 rows. Figure 2 shows the source database installed in SQL Server 2000. Figure 3 shows the base table in this database.

We have implemented the bucket algorithm and containment bucket algorithm and compared the performance of the both algorithm. To facilitate the experiments, we implemented a random view generator which enables us to control the parameter which is the number of views. The views used in the experiments were randomly generated select-project-join views over the SQL SEVER 2000 database. Each view was generated by randomly selecting a base table and joining in additional base tables. The number of views is the parameter of this views generator. The number of selected columns for each view is controlled less than ten. An important variable to keep in mind throughout the experiments is the number of potential rewritings that can be generated.

[4] http://www.flcdatacenter.com/CaseData.aspx

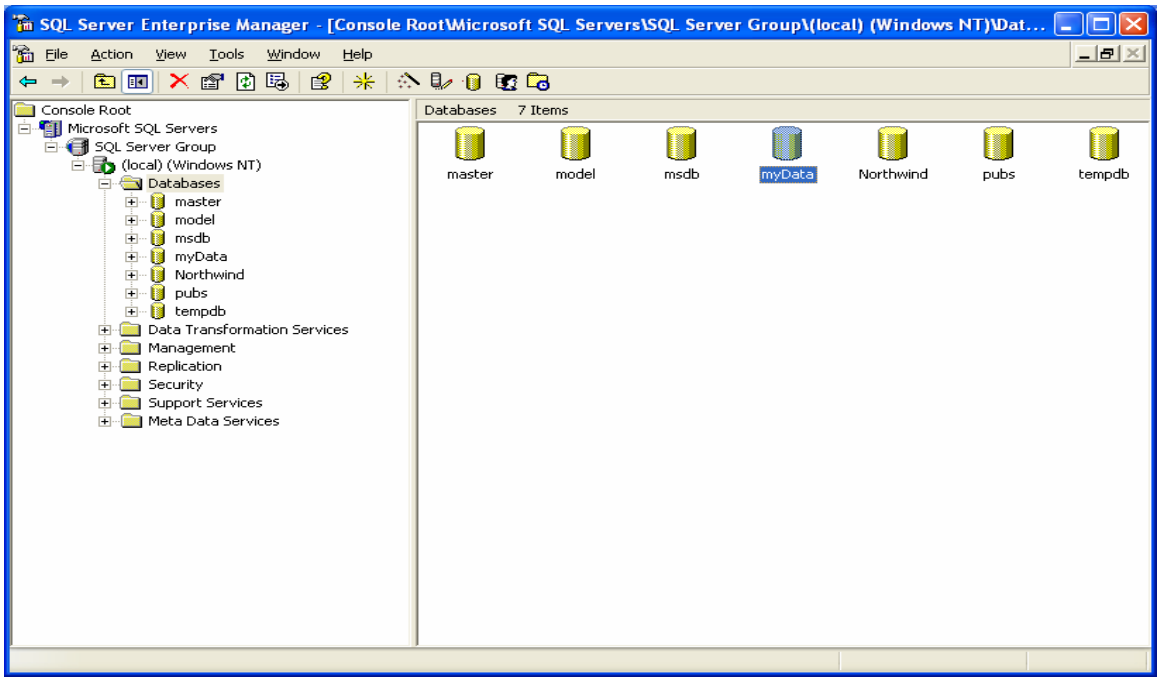

Figure 2: Source database installed in SQL Server 2000

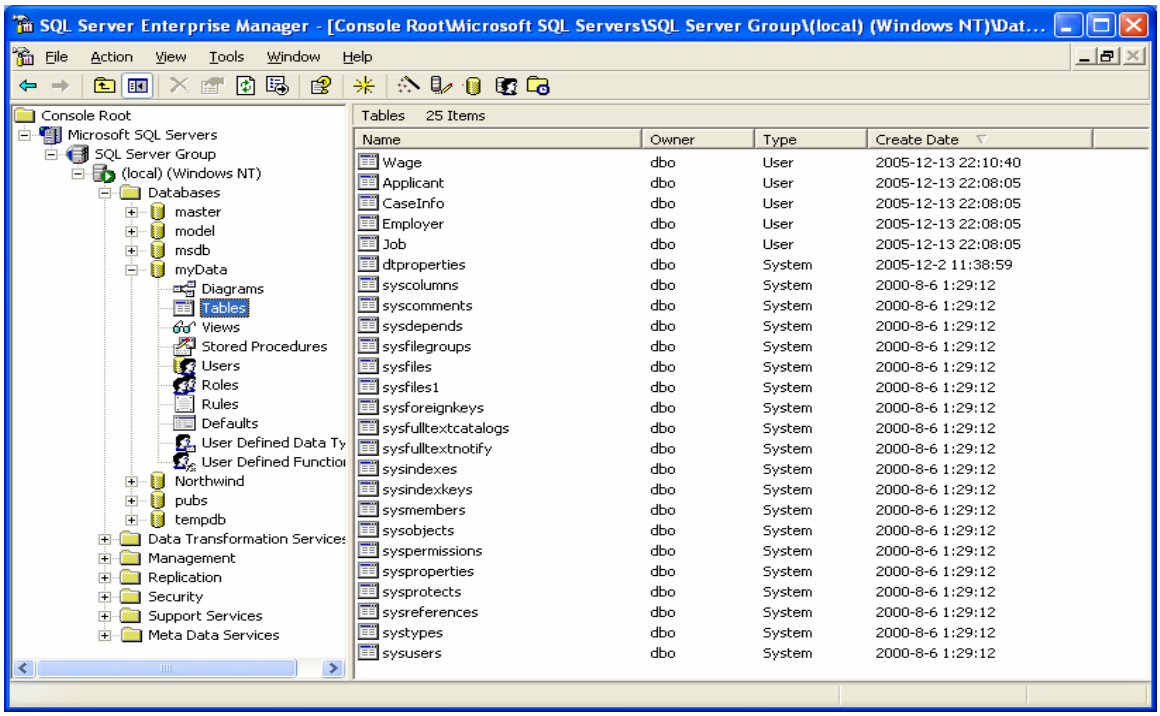

Figure 3: Base tables of the source data in SQL Sever 2000

In the first case, all the variables in the views are randomly selected, shown in figure 4. The bucket algorithm performs much worse than containment bucket algorithm, because of the number and cost of the query containment checks which is in second phase it needs to perform. The containment bucket algorithm outperforms the bucket algorithm significantly. However, the variance in the results generated by bucket algorithm is very high because some of the queries in the experiment have a huge number of potential rewritings which take much more time, while others have a very small number of potential rewritings.

Figure 5 shows the containment bucket algorithm in detail.

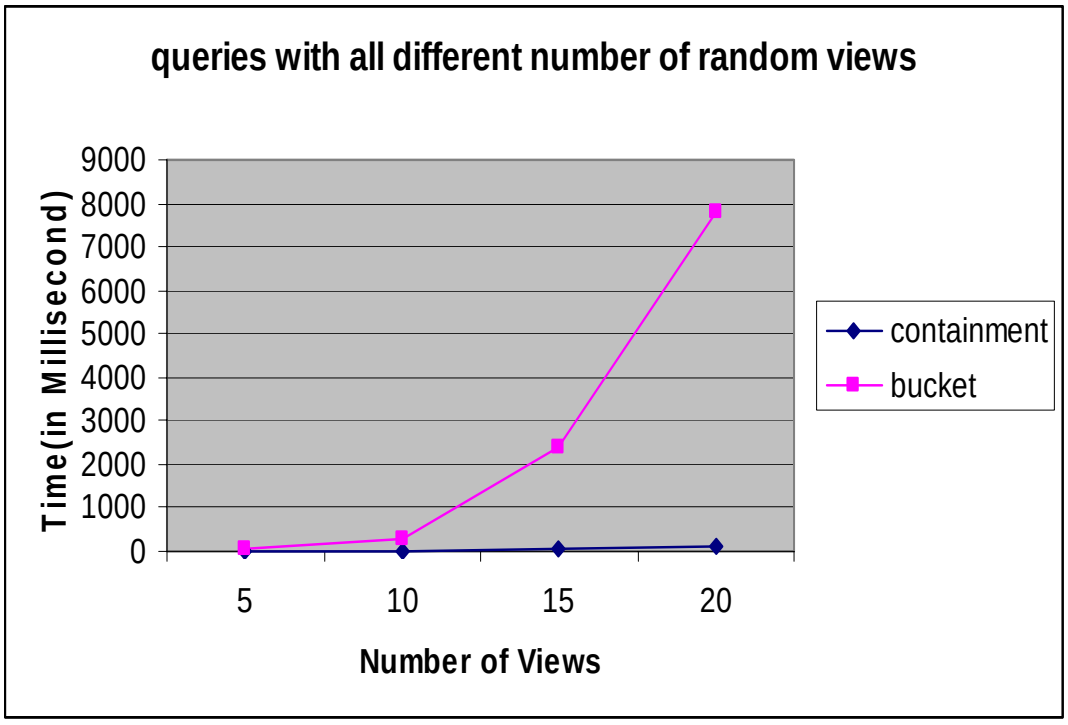

Figure 4: Running time for query writing by both algorithms

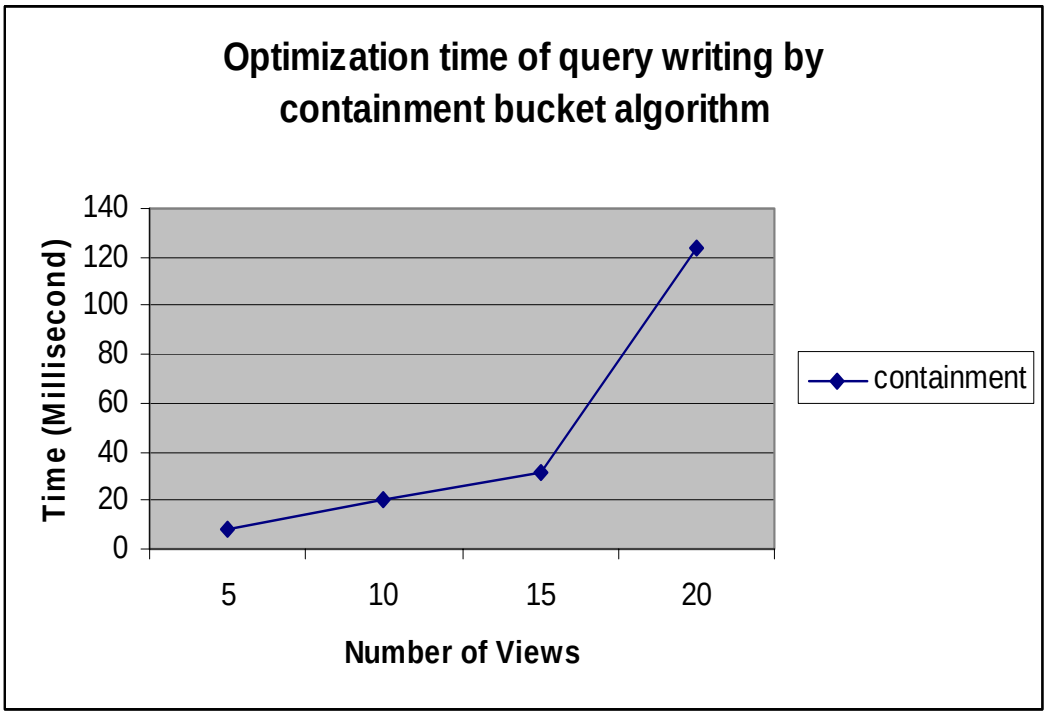

Figure 5: Running time for query writing by containment bucket algorithm

The difference in the performance between the bucket algorithm and the containment bucket algorithm in this context is due to the second phases of the algorithms. In the second phase, the bucket algorithm is searching for all candidate rewriting queries generated by executing a Cartesian product of the views in the bucket, then compares all queries to find a best one with lowest number of views involved and with all query head variables included. Thus, the second step becomes very expensive. The containment bucket algorithm is searching a much smaller space, because the number of views in the subgoal buckets is smaller than the number of views in the subgoal buckets generated by bucket algorithm. Moreover, the containment bucket algorithm is performing better because in the first phase of the algorithm it already removed from consideration views that may not be usable due to the views are not containment target views for the user query.

The amount of work that the bucket algorithm will waster depends on when it will remove unrelated views. The first phase considers the subgoals in the query when it put all related views in the bucket. If a failure appears late in the process, more work is wasted. The important point is that the optimal order in which to consider the subgoal depends heavily on the specific views available.

We also consider another case. We put a view manually into database, this view has to be identical to the query, and as a result there are very few rewritings. The graph in Figure 6 shows that on average the containment bucket algorithm performs better than the bucket algorithm by anywhere which is same as the first case. It this case, bucket algorithm still generates many rewritings and hence the performance of the algorithms is limited because of the sheer number of rewritings. Actually, in this case, the containment bucket algorithm finds the candidate solution after it scans the whole views in the first phase, since every subgoal bucket got the same view which is identical to the query, so that in the second phase, we only need to pick up the view quickly from each bucket, and get the rewrite query.

Figure 7 shows the performance of first case and second case generated by bucket algorithm. There is no much difference between these two cases. The algorithm is forced to form a possibly exponential number of rewritings, for the queries and views with five subgoals, the bucket algorithms take on the number of candidate solutions of more than 200 for give five views.

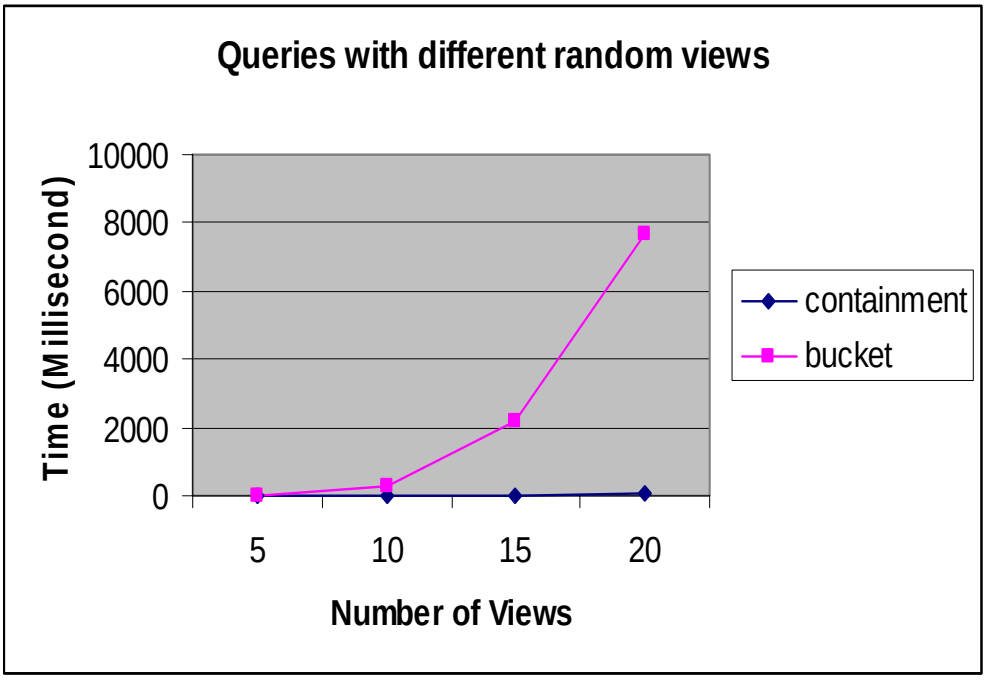

Figure 6: Running time for query writing by both algorithms

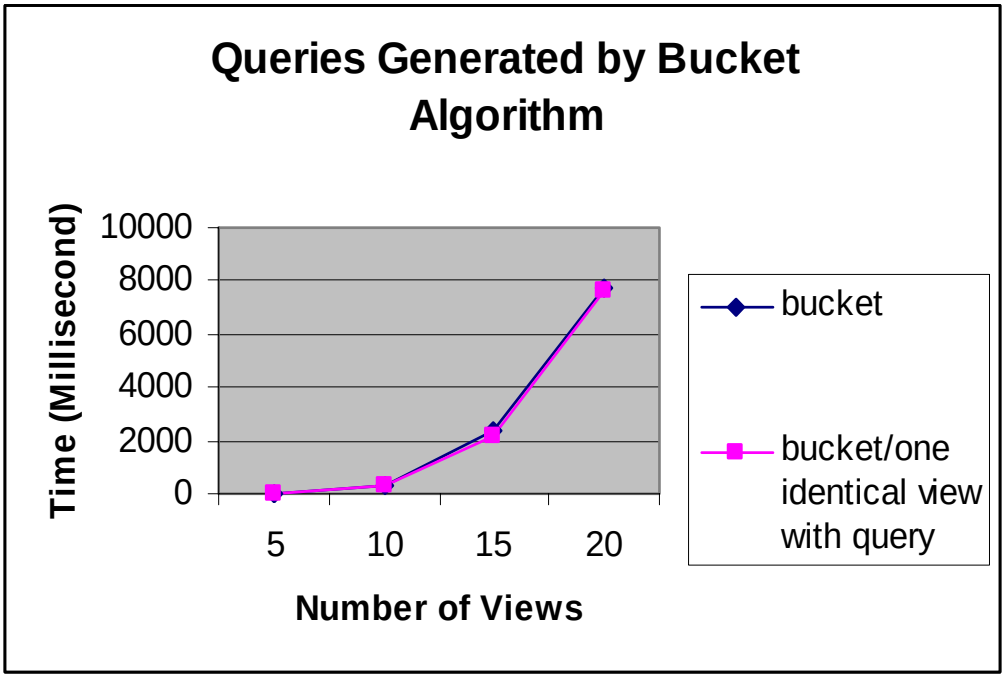

Figure 7: Running time for query writing by Bucket Algorithm

The comparison between first case and second case is generated by containment bucket algorithm shown Figure 8. The containment bucket algorithm performs in first case worse than in the second case, because of the number of views put in bucket of subgoals is smaller than those in the first case. Thus, in the second phase, the algorithm generates very smaller number of combinations which are potential query rewritings.

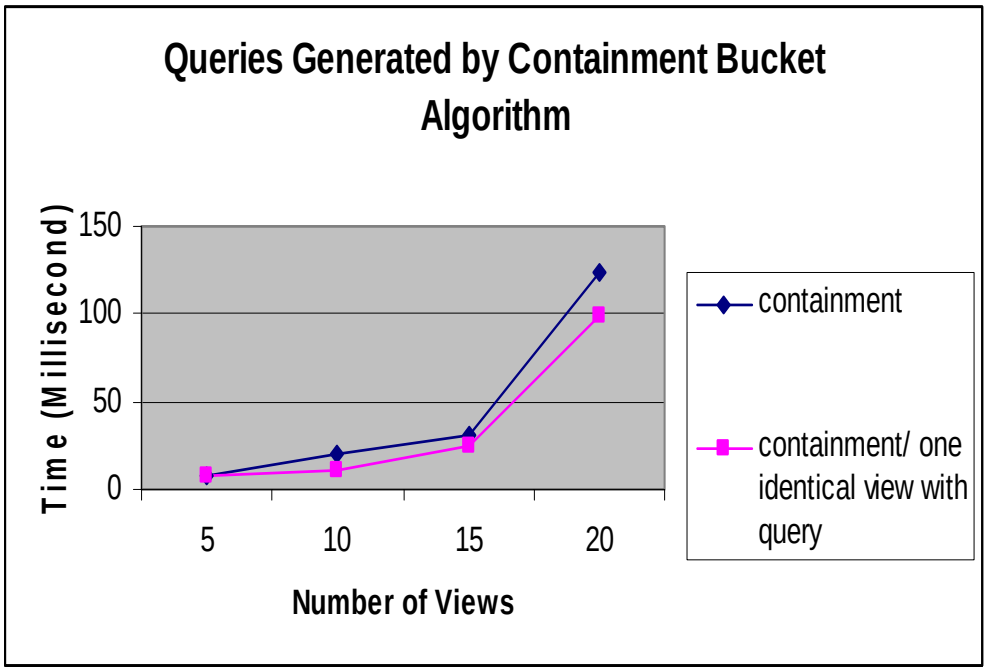

Figure 8: Running time for query writing by Containment Bucket Algorithm

The experiments showed that the saving for the containment bucket algorithm over the bucket algorithm, as expected, grew with the number of views and the number of subgoals in the query; this is because the number of combinations that was considered was much higher and thus the smaller search space that the containment bucket algorithm considered was much more evident.

CHAPTER VII

CONCLUSION AND FUTURE WORKS

A data warehouse is a user-centered environment for data analysis and decision support. To support decision maker to make decision quickly and accurately, using materialized views can provide massive improvements in query processing time. This thesis presents a new algorithm for answering queries using materialized views. Materialized views can reduce query processing time very significantly but only if the query optimizer is able to find applicable views quickly. We began by analyzing the existing algorithm: bucket algorithm, and found that it has significant limitation. We developed the containment bucket algorithm, a better algorithm for answering queries using views and it outperforms the existing algorithm. The experiments showed that the bucket algorithm performed worse than containment bucket algorithm in both cases. In both cases the containment bucket algorithm outperformed the bucket algorithm by differing factors. The problem with the bucket algorithm is that discovers many of the interactions between the views in its second phase, and the performance in that phase is heavily dependent on the number of views in each query subgoal. Since performance depends heavily on the interaction with the views, the containment bucket algorithm gives optimal method to solve this problem. We developed the containment algorithm for answering queries using views and showed outperforms the bucket algorithms.

The following issues will be focused on in the future work:

- Make containment algorithm scales to very large number of views very well. We will use large number of views to compare both algorithms. Based on our current experiments, the larger number of views, the better performance of containment bucket algorithm.
- Extend to handle comparison predicates and show its performance experimentally. In this thesis, we only consider the class of select-project-join queries. However, in the really data warehouse environment, the comparison predicates are also common occurred in the views [CAN99]. After we solve this problem, the containment bucket algorithm will be accepted widely.

REFERENCES

[AD98] Abiteboul, S., Duschka, O., *Complexity of answering queries using materialized views.* In: Proc. of Principles of Database Systems, PP 254-263, Seattle, Wash., USA 1998

[AGK99] Afrati, F., Gergatsoulis, M., Kavalieos, T, *Answering queries using materialized views with disjunctions.* In: Proc. of International Conference on Database Theory, PP435-452, 1999

[CBC05] Connolly, T. M., Begg, Carolyn E.: *Database Systems: a practical approach to Design, Implementation, and Management*, Fourth Edition, Addison-Wesley, 2005

[CD97] Chaudhuri, S., Dayal, U.: *An Overview of Data Warehousing and OLAP Technology*. ACM SIGMOD Record, 26(1):65--74, March 1997

[CL03] Chirkova, R., Li, C., *Materializing views with Minimal size to answer queries*, In PODS, 2003, June 9-12.

[CKPS95] Chaudhuri, S., Krishnamurhy, S., Potamianos, S., Shim, K., *Optimizing Queries with Materialized views*, International Conference on Data Engineering, 1995.

[CAN99] Cohen, S., Nutt, W., Serebrenik, A., *Rewriting Aggregate Queries Using Views*, In: Proc. of Principles of Database Systems, 1999.

[CL90] Chirkova, R., Li, C., *Materializing views with Minimal size to answer queries*, In: Proc. of Principles of Database Systems, 2003, June 9-12.

[D00] Date, C. J., *An Introduction to Database Systems, Seventh Edition*, Addison-Wesley, pp. 292-293 2000.

[DG97] Duschka, O. M., Geneserech, M. R., *Answering recursive queries using views*, In: proc. of the 16th ACM Symposium on Principle of Database Systems, Tucson, AZ, pages 109-116, 1997.

[G03] Gursahani, A., *Materialized Views in Oracle*, In: Database Journal, from http://www.databasejournal.com/features/oracle/article.php/2192071, 2006,

February.

[H00] Halevy, A. Y., *Theory of Answering Queries Using Views*, In: Special Interest Group on Management of Data Record, Vol. 29, No. 4, December 2000

[H01] Halevy, A.Y. *Ansering queries using views*: *A survey*, In: Very Large Databases, 2001.

[JLVV00] Jarke, M., Lenzerini, M., Vassiliou, Y., Vassiliadis, P.: *Fundamentals of Data Warehouses*, W. springer-Verlag Berlin Heidelberg 2000, ISBN 3-540-65365-1

[LMSS95] Levy, A. Y., Mendelzono, A. O., Sagiv, Y., Srivastava, D., *Answering queries using views*, In: Proc. of Principles of Database Systems, pp. 95-104, San Jose, Calif, USA 1995

[LR99] Li, Z., Ross, K. A., *Fast joins using join indices*, In: Very Large Databases, 1999, 8: 1-24

[LRO96] Levy, A. Y., Rajaraman, A., Ordille, J. O., *Querying herterogeneous information sources using source descriptions*, In: Very Large Databases, pp. 251-262, 1996.

[MW04] Morzy, T., Wrembel, R., *On Querying Versions of Multiversion Data Warehouse*, In: International Workshop on Data Warehousing and OLAP 2004, November, 92-101

[QL01] Qiu, S. G., Ling, T. W., *Index Filtering and View Materialization in Relational OLAP Environmnet*, Conference on Information and Knowledge Management ’01, November -10, 2001

[RBN02] Rifaieh, R., Benharkat, N. A.: *Query-based Data Warehousing Tool.* DOLAP ’02, November 8, 2002, Mclcan, Virginia, USA

[SB00] Saharia, A. N., Babad, Y. M., *Enhancing Data Warehouse Performance through Query Caching*, The data base for Advances in Information Systems, Summer 2000. Vol. 31, No. 3

[SWS02] Stockinger, K., Wu, K., Shoshani, A., *Strategies for Processing ad hoc Queries on Large Data Warehouses*, In: International Workshop on Data Warehousing and OLAP 2002, November 8, 2002, Mclean, Virginia, USA.

[U88] Ullman, J. D., *Principles of Database and Knowledge-base Systems, Volume II*, Computer Science Press, pp. 877-916, 1988.

[VS99] Vassiliadis, P., Sellis, T., *A survey of logical models for OLAP databases* Special Interest Group on Management of Data, Record, Volume 28 Issue 4.

APPENDIX

Program Code

```
/*****************************************************************
 * Author Name: Jing Hu
 * Date: 1-4-2006
 * Advisor: Dr. G. E. Hedrick
 * Title: Optimizing queries using materialized view in
 *           data warehouse--Master Thesis Experimentation
 * This program is written in Java language.
 * Using JDBC connnect to MS SQL Server 2000.
 * This database is from immigration goverment website
 *
 * This program is designed to compare the performance of
 *   of two query writing using materilized views.
 *
 *
 *****************************************************************/

import java.io.*;
import java.io.IOException;
import java.io.InputStreamReader;
import java.sql.*;
import java.util.*;

/*******************
 * Class Bucket
 *******************/
public class Bucket {

      public static Vector queryHead = new Vector(1);
                        // store all heads of user query in this vector
      public static Vector queryTable = new Vector(1);
                        // store all tables involved the user query
      public static int numberOfViews = 0; //number of views in database
      public static Vector[] tableCol;
                        // store table columns involved views,
      public static Vector viewNameVec = new Vector(1);
                        // store view name
                        // index of tableCol and viewNameVec are
```

```
        corresponding

    public static Vector[] viewTableVec;
                    // store table name involved in each view
    public static Vector[] viewColVec;
                    // store columnn name involved in each view

    public static Vector[] viewSelect;
                    // store select part in every table involved each view

    public static Vector tableNameVec = new Vector(1);
                    // store all base table names
    public static Vector[] tableColumn;
                    //table column names involved in each tables

    //public static String bucketQuery = "";
    //public static String myQuery = "";

/*********************************************
 * The main method is to call other methods.
 *********************************************/
    public static void main(String args[]) {

        String url = "jdbc:odbc:myData";
        String query = "";
        Connection con;
        try
        {
            FileReader file = new FileReader("query.txt");
            BufferedReader stdin = new BufferedReader(file);
              query = stdin.readLine();
        }
        catch(IOException e){
          }
          /* query = "SELECT Name, JobTitle, DesignatedFirstName, " +
                "DesignatedLastName " +
                "FROM Employer, Job, Applicant, Wage, CaseInfo " +
                "WHERE Employer.CaseNo=Job.CaseNo AND " +
                "Job.CaseNo=Applicant.CaseNo "+
                "AND Wage.CaseNo=CaseInfo.CaseNo AND " +
                "Applicant.CaseNo=Wage.CaseNo "+
                "and CaseInfo.ApprovalStatus = 'Certified' " +
                "and Wage.WageRate1 > 50000";*/


        setQuery(query);
        // set user query, get value for queryHead and queryTable

        Statement stmt;
```

```
try
{
     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          // setup JDBC Driver

} catch(java.lang.ClassNotFoundException e) {
     System.err.print("ClassNotFoundException: ");
     System.err.println(e.getMessage());
}

try
{
     con = DriverManager.getConnection(url,    "sa", "");
          // connect to database

     // get tables information
     stmt = con.createStatement();
     ResultSet rsTables = stmt.executeQuery("SELECT * FROM" +
                         " Information_Schema.Tables");
                    // execute this query to get table information

     setTables(rsTables);// get tableNameVec value
     stmt.close(); // close this statement

     tableCol = new Vector [queryTable.size()];
                    // initialize table column size which is same
                    // as the size of queryTables

     for (int i = 0; i < queryTable.size(); i++){
          tableCol[i] = new Vector();
          // initialize the tableCol, the size is same as queryTable
     }

     int numberOfTables = tableNameVec.size();

     tableColumn = new Vector[numberOfTables];
                    // initialize tableColumn
     for (int i = 0; i < numberOfTables; i++)
          tableColumn[i] = new Vector();

     // get table columns information
     stmt = con.createStatement();
     ResultSet rsTableColumns = stmt.executeQuery("SELECT * FROM" +
                         " Information_Schema.Columns");

     setSubgoalHead(rsTableColumns);
          // get tabelNameVec and tableName values

     stmt.close(); // close current statement
```

```
// create views
stmt = con.createStatement(); // open a statement
int viewNumber = 30; // set the number of views
      // it is parameter of random view generator

String createViewString = "";
      // temp string of storing SQL statement for creating views

// number of loops equal the number of random views created
for (int i = 30; i < viewNumber; i ++){

      createViewString = "CREATE VIEW VIEW" + i + setViews();
      stmt.execute(createViewString);
            // execute the SQL statement for create views
}

stmt.close();//close current statement

stmt = con.createStatement();

// get views information:
ResultSet rsViews = stmt.executeQuery("SELECT * " +
                              "FROM Information_Schema.Views");
ResultSetMetaData rsmdViews = rsViews.getMetaData();

int numberOfColumnsR = rsmdViews.getColumnCount();

String viewDefinition = "", tableName1= "";
Vector selectVar = new Vector();
            // store select variables from string to vector

while (rsViews.next())
{
      tableName1 = rsViews.getString("TABLE_NAME");

            viewDefinition = rsViews.getString("VIEW_DEFINITION");

      if (!tableName1.equals("syssegments")&&
                  !tableName1.equals("sysconstraints"))
                        // remove non table name
      {
            numberOfViews++;    //count number of views
            selectVar.addElement(setViewSelect(viewDefinition));
                        // convert string to vector
            if(!viewNameVec.contains(tableName1))
            {           // check if viewNameVec contains viewName
                  viewNameVec.addElement(tableName1);
                        // store view names into vector
            }
```

```
                }
            }

            viewTableVec = new Vector [numberOfViews];
                        //set table array in view
            viewColVec = new Vector [numberOfViews];
                        //set column array in view
            viewSelect = new Vector [numberOfViews];
                        //set select array in view

            //initialize these variables
            for (int i = 0; i < numberOfViews; i++){
                viewTableVec[i]= new Vector();
                viewColVec[i]= new Vector();
                viewSelect[i] = new Vector();
            }

            StringTokenizer tk;

            // set viewSelect value
            for ( int i = 0; i < numberOfViews; i++)
            {
                tk = new StringTokenizer(selectVar.elementAt(i).toString());

                while (tk.hasMoreTokens())
                {
                    String tttt = tk.nextToken().trim();
                    viewSelect[i].addElement(tttt);
                }
            }


            stmt.close();//close current statement.

            stmt = con.createStatement();  //open a new statment

            // get column of views information
            ResultSet rsRView = stmt.executeQuery("SELECT * FROM" +
                        "
INFORMATION_SCHEMA.VIEW_COLUMN_USAGE");

            getViews(query, rsRView);

            stmt.close(); //close the statement

            con.close(); //close the connection

        } catch(SQLException ex) {
            System.err.println("SQLException: " + ex.getMessage());
        }
```

```
            compareAlg(); // compare two algorithms
        }

/**********************************
 * compare two alg's running time
 * ********************************/
        public static void compareAlg(){

            long startTimeBucket, stopTimeBucket, startTimeCon, stopTimeCon;
            long bucketRunTime, containmentRunTime;

            startTimeCon = System.currentTimeMillis();
            String containmentString = containmentBucketRewriting();
            System.out.println("containmentString-> "+ containmentString);
            stopTimeCon = System.currentTimeMillis();
            containmentRunTime = stopTimeCon - startTimeCon;

            System.out.println("containmentRunTime-> "+ containmentRunTime);

            startTimeBucket = System.currentTimeMillis();

            String bucketString = bucketRewriting();
            System.out.println("bucketString-> "+ bucketString);
            stopTimeBucket = System.currentTimeMillis();

            bucketRunTime = stopTimeBucket - startTimeBucket;
            System.out.println("bucketRunTime-> "+ bucketRunTime);
        }

/*****************************************
 *    Find all base tables and store them
 *    to a vector tableNameVec
 *****************************************/
        public static void setTables(ResultSet rs){

            String tableName = "", tableType = "";
            int curTableIndex = 0;

            try
            {
                while (rs.next()) {

                    tableName = rs.getString("TABLE_NAME");
                                // get table name in view

                    tableType = rs.getString("TABLE_TYPE");
                                // get column name in view

                    if(tableType.equals("BASE TABLE"))
```

```
                    tableNameVec.addElement(tableName);
                        // add table name to a vector
                }

                tableNameVec.remove("dtproperties");
                        // remove the dtproperties table

            }
            catch(SQLException ex) {
                System.err.println("SQLException: " + ex.getMessage());
            }

            //System.out.println("tablenamevec2-> ");printVector(tableNameVec);
        }

/***********************************************
 * Creat random views
 * Return a string: part of SQL statement
 ***********************************************/
    public static String setViews(){

        String createViewString = "", tempStatement = "";
        String columnName = "", tableName = "";
        Vector tableNameVecLocal = new Vector();
                // store table name for From part
        Vector columnNameVecLocal = new Vector();
                // store select column names with table name for select part.
        Vector columnNameVecTemp = new Vector();
                // store select column without table names

        Random generatorTable, generatorCol;
        generatorTable = new Random();
                // generate the number of tables used in views
        generatorCol = new Random();
                // generate the number of select columns in each view
        int columnSize = 0, colNo = 0, r = 0 ;
        r = generatorTable.nextInt(tableNameVec.size()) + 1;
                // get number of tables
        String columnTemp = "", columnNoTName = "";

        for ( int i = 0; i < r; i ++)
        {
            tableNameVecLocal.addElement(tableNameVec.elementAt(i));
            columnSize = tableColumn[i].size();
            colNo = generatorCol.nextInt(columnSize) + 1;
                    //get number of column in each table

            for ( int t = 0; t < colNo; t++){

                columnNoTName = tableColumn[i].elementAt(t).toString();
```

```
                    if(columnNameVecTemp.size() < 10 &&
                            // let select part size less than 10
                        !columnNameVecTemp.contains(columnNoTName)){
                        columnNameVecTemp.addElement(columnNoTName);
                            // add column without table name to a vector

                        columnTemp = tableNameVec.elementAt(i).toString()+
                            "."+tableColumn[i].elementAt(t).toString();
                            // add corrosponding column with table name

                        columnNameVecLocal.addElement(columnTemp);
                            // add corrosponding column with table name to a
                            vector
                    }
                }

                tableName = tableNameVecLocal.toString();
                        // convert vector to string
                tableName = tableName.substring(1, tableName.length()-1);
                        //remove    []
                columnName = columnNameVecLocal.toString();
                        // convert vector to string
                columnName = columnName.substring(1, columnName.length()-1);
                        // remove []
                tempStatement =    " AS SELECT " + columnName + " FROM " +
                tableName;
            }

            createViewString = tempStatement;
            return createViewString;

        }// end setViews method

/**********************************************************
 * Containment Bucket Algorithm
 * Return a string which is a rewriting query using views
 * ********************************************************/
        public static String containmentBucketRewriting(){

            Vector subgoal = new Vector(1);
            Vector [] bucket ;
            String subgoalTemp ="";
            int numberOfSubgoal = 0, viewNumber = 0,
                smaller = 1000, index = 0;

            numberOfSubgoal = queryTable.size();

            bucket = new Vector[numberOfSubgoal];
                    // create bucket
```

```
//initialize bucket
for (int i = 0; i < numberOfSubgoal; i++)
{
      bucket[i]= new Vector();
}

//Create subgoal now
for (int i = 0; i<numberOfSubgoal; i++)
{
      subgoal.addElement(queryTable.elementAt(i));
}


//fill buckets based on columns
for (int i = 0; i < numberOfSubgoal; i++){//subgoal/query table

      subgoalTemp = subgoal.elementAt(i).toString();

      for (int t = 0; t < viewSelect.length; t ++)//view head columns
      {
            if (checkQueryHead(tableCol[i], viewSelect[t])
                        //true if viewSelect contains tableCol
                  && checkBucket(subgoalTemp,bucket[i])
                        //true if bucket doesnt contain subgoaltemp
                  && viewTableVec[t].contains(subgoalTemp))
                        // no depulicate table name added
            {
                bucket[i].addElement(viewNameVec.elementAt(t));
            }
      }
}
Vector pQueryVec = new Vector(1);

//get cartesian product of the buckets
pQueryVec = cartPro(subgoal,bucket);

// pick rewrite query with smallest number of views
if(pQueryVec.size() > 1 )
{
      for (int t = 0; t < pQueryVec.size(); t ++)
      {
            viewNumber = pQueryVec.elementAt(t).toString().length();
            if (viewNumber < smaller )
            {
                  smaller = viewNumber;
                  index = t;
            }
      }
}
```

```
        String queryView = "";
        queryView = pQueryVec.elementAt(index).toString();
        queryView = queryView.substring(1,queryView.length()-1);
                            // remove []
        String queryH = queryHead.toString();
        queryH = queryH.substring(1, queryH.length()-1);

        String fromStatement = "", tempView1 = "", rewrQuery = "";

        StringTokenizer queryV = new StringTokenizer(queryView, ",");
        Vector viewNameVec1 = new Vector();//get new clear vector

        //set query head from string to vector
        while (queryV.hasMoreTokens())
        {
            viewNameVec1.addElement(queryV.nextToken());
        }

        queryH = getSelectWithViewName(viewNameVec1, queryH);

        fromStatement = setFromStatement(queryView);
        rewrQuery = "SELECT " + queryH + " FROM " + fromStatement;
        return rewrQuery;
    }

/*****************************************************************
 * get view vector pViews and query head to generate the query
 * select part, return a string with viewName.column
 * ***************************************************************/
    public static String getSelectWithViewName(Vector pViews, String queryHe){

        String strReturn = "";
        StringTokenizer queryTK, viewTK;
                //set query head select parts to stringtokenizer
        Vector queryHeadT = new Vector();
        String stringTK ="", tempSt ="", pViewStr = "", headTemp = "";
        int indexOfView = 0;

        Vector pViewVec = new Vector();
                // put all views in potiential solution in one vector
        Vector[] pViewSelect ;
                // put all selects in p s in one vector.
        Vector tempSelectWViewName = new Vector();
                // store temp vector for query head with view name

        pViewStr = pViews.toString().trim();
        pViewStr = pViewStr.substring(1, pViewStr.length()-1);
                // remove []
        viewTK = new StringTokenizer(pViewStr, ",");
```

```
        //set all views in vector pViewVec.
        while (viewTK.hasMoreTokens())
        {
            String temptemptemp = viewTK.nextToken().trim();
            pViewVec.addElement(temptemptemp);
        }

        pViewSelect = new Vector[pViewVec.size()];//initial pViewSelect

        for (int i = 0; i < pViewVec.size(); i ++)
        {
            pViewSelect[i] = new Vector();//initial pViewSelect
            indexOfView = viewNameVec.indexOf(pViewVec.elementAt(i));
            pViewSelect[i] = viewSelect[indexOfView];
                //set select parts corrosponding selected views
        }

        queryTK = new StringTokenizer(queryHe, ",");
        String tempStore = "";

        while (queryTK.hasMoreTokens()){

            tempSt = queryTK.nextToken().trim();

            for(int i = 0; i < pViewSelect.length; i++){

                if(pViewSelect[i].contains(tempSt)
                && !tempSt.equals(tempStore) ){

                    headTemp = pViewVec.elementAt(i) + "." + tempSt;
                    tempSelectWViewName.addElement(headTemp);
                    tempStore = tempSt;
                }
            }
        }
        strReturn = tempSelectWViewName.toString();
        strReturn = strReturn.substring(1, strReturn.length()-1);

        return strReturn;
    }

/**********************************************************
 * bucket Algorithm
 * return string: a rewritten query by bucket algorithm
 **********************************************************/
    public static String bucketRewriting(){

        Vector subgoal = new Vector(1);
        Vector [] bucket ;
```

```
String rewrQuery = "", onePViews = "", tempView = "",
        tempSelect = "", headSelectPartStr="";
int numberOfSubgoal = 0,indexOfView = 0;
numberOfSubgoal = queryTable.size();
StringTokenizer tkn;

bucket = new Vector[numberOfSubgoal];

//initialize bucket
for (int i = 0; i < numberOfSubgoal; i++)
{
    bucket[i]= new Vector();
}

//Create subgoals now
for (int i = 0; i<numberOfSubgoal; i++)
{
    subgoal.addElement(queryTable.elementAt(i));
}

String subgoalTemp = "";// tableTemp ="";

//fill buckets
for (int i = 0; i < numberOfSubgoal; i ++)
{
    subgoalTemp = subgoal.elementAt(i).toString();

    for(int t = 0; t < viewNameVec.size(); t ++)
    {
        if (viewTableVec[t].contains(subgoalTemp)
                && checkBucket(subgoalTemp,bucket[i]))
        {
            bucket[i].addElement(viewNameVec.elementAt(t));
        }
    }
}

Vector pQueryVec = new Vector(1);

//get cartesian product of the buckets
pQueryVec = cartPro(subgoal,bucket);

//pick one query views from all potiential queries

Vector allSelectPart = new Vector();
Vector headSelectPart = new Vector();
        //removed all potiential queries which donot show on queryhead
Vector headSelectView = new Vector();
        //views corrosponding select part.
```

```
        for (int i = 0; i < pQueryVec.size(); i++)
        {
            onePViews = pQueryVec.elementAt(i).toString();
            onePViews = onePViews.substring(1,onePViews.length()-1);
                                //remove [] in string
            tkn = new StringTokenizer(onePViews,",");

            while(tkn.hasMoreTokens()){

                tempView = tkn.nextToken().trim();
                indexOfView = viewNameVec.indexOf(tempView);
                                //get index of same view

                for (int t = 0; t<viewSelect.length; t++)
                {
                    for(int n = 0; n<viewSelect[t].size(); n++)
                    {

                    tempSelect = viewSelect[t].elementAt(n).toString().trim();
                    if ( !allSelectPart.contains(tempSelect))
                    {
                        allSelectPart.addElement(tempSelect);
                    }
                    }
                }
            }

            if (checkQueryHead(queryHead,allSelectPart))
            {
                headSelectPart.addElement(headSelectPartStr);
                headSelectView.addElement(onePViews);
            }

            headSelectPartStr = allSelectPart.toString();
            headSelectPartStr =
headSelectPartStr.substring(1,headSelectPartStr.length()-1);

        }

        int index = 0, viewNumber = 0, smaller =1000;

        //pick rewrite query with smallest number of views
        for (int t = 0; t < headSelectPart.size(); t ++)
        {
            viewNumber = headSelectView.elementAt(t).toString().length();
            if (viewNumber < smaller )
            {
                smaller = viewNumber;
                index = t;
            }
```

```
        }

        //create query string
        String queryView = headSelectView.elementAt(index).toString();
        String queryH = queryHead.toString();
        queryH = queryH.substring(1, queryH.length()-1);

        Vector queryViewVec = new Vector();
        StringTokenizer queryHTK = new StringTokenizer(queryView, ",");

        //set query head from string to vector
        while (queryHTK.hasMoreTokens())
        {
            queryViewVec.addElement(queryHTK.nextToken());
        }

        String fromStatement = "", tempView1 = "";

        fromStatement = setFromStatement(queryView);

        queryH = getSelectWithViewName(queryViewVec, queryH);

        rewrQuery = "SELECT " + queryH + " FROM " + fromStatement;
        return rewrQuery;
    }

/*****************************************************************
 * set from part statement: plus "inner join...on..."
 * parameter@: fromStr is the string which includes all views
 * return a string which includes whole part of from statement
 *****************************************************************/
    public static String setFromStatement (String fromStr){

        String returnStr = "", tempView1 = "", tempView2 = "",
                tempView3 = "", fromStatement="";
        StringTokenizer queryV = new StringTokenizer(fromStr, ",");

        tempView1 = queryV.nextToken();
        fromStatement += tempView1;

        // convert "from" part
        while(queryV.hasMoreTokens())
        {
                tempView2 = queryV.nextToken();
                tempView3 = tempView1;
                fromStatement += " INNER JOIN " + tempView2 + " ON "
                + tempView3 + ".CaseNo = "+ tempView2 +".CaseNo ";
                tempView1 = tempView2;
        }
        returnStr = fromStatement;
```

```
        return returnStr;
    }

/************************************************
 * set view names, tables, columns
 * get query string q, and resultset rs
 ************************************************/
    public static void getViews(String q, ResultSet rs){

        String tableName = "", viewName = "", columnName = "";
        int curViewIndex = 0;

        try
        {
            while (rs.next()) {

                viewName=rs.getString("VIEW_NAME");
                            // get view name in view

                tableName = rs.getString("TABLE_NAME");
                            // get table name in view

                columnName = rs.getString("COLUMN_NAME");
                            //get column name in view

                if ( !viewName.equals("sysconstraints"))// skip non view name
                {
                    curViewIndex = viewNameVec.indexOf(viewName);

                    if(checkBucket(tableName, viewTableVec[curViewIndex]))
                                        // remove duplicated tables
                    {
                        viewTableVec[curViewIndex].addElement(tableName);
                    }
                    viewColVec[curViewIndex].addElement(columnName);
                }
            }
        }
        catch(SQLException ex) {
            System.err.println("SQLException: " + ex.getMessage());
        }
    }

/*******************************************
 * set table columns
 *******************************************/
    public static void setSubgoalHead(ResultSet rs){

        String tableName = "", columnName = "";
        int curTableIndex1 = 0, curTableIndex2 = 0;
```

```
            try
            {
                while (rs.next())
                {
                    tableName = rs.getString("TABLE_NAME");
                                // get table name in view
                    columnName = rs.getString("COLUMN_NAME");
                                // get column name in view

                    if( queryTable.contains(tableName)){
                        curTableIndex1 = queryTable.indexOf(tableName);

                        if(checkBucket(columnName, tableCol[curTableIndex1]))
                                //remove duplicated tables
                        {
                            tableCol[curTableIndex1].addElement(columnName);
                        }
                    }

                    //set columns based on table
                    if( tableNameVec.contains(tableName)){
                        curTableIndex2 = tableNameVec.indexOf(tableName);

                        if(checkBucket(columnName, tableColumn[curTableIndex2]))
                                //remove duplicated tables
                        {
                            tableColumn[curTableIndex2].addElement(columnName);
                        }
                    }
                }
            }
            catch(SQLException ex) {
                System.err.println("SQLException: " + ex.getMessage());
            }
        }

    /************************************************************
     * check if viewVec includes all queryVec return true
     * ************************************************************/
        public static boolean checkQueryHead (Vector queryVec, Vector viewVec){

            boolean flag = true;

            for (int i = 0; i < queryVec.size(); i ++)
            {
                if (!viewVec.contains(queryVec.elementAt(i)))
                {
                    flag = false;
                    break;
```

```
                }
            }
            return flag;
        }

/*************************************************************
 * set view SELECT part
 * parameter @ str: whole query string
 * return string which is only SELECT part
 *************************************************************/
        public static String setViewSelect(String str){

            String selectPart = "", tempSt = "", reStr = "";
            selectPart = str.substring(str.indexOf("SELECT")+7,
                                       str.indexOf("FROM")-1);

            StringTokenizer tk = new StringTokenizer(selectPart, ",");

            while ( tk.hasMoreTokens() )
            {
                tempSt = tk.nextToken();
                tempSt = tempSt.substring(tempSt.indexOf(".")+1);
                        // get substring from first .
                tempSt = tempSt.substring(tempSt.indexOf(".")+1);
                        // get substing from 2nd .
                tempSt = tempSt + " ";
                reStr += tempSt;
            }

            return reStr;
        }

/*************************************************************
 * Cartesian product of the buckets
 * Parameter @ subgoalVec: includes all subgoal
 * Parameter @ bucketVecArr: includes all buckets
 *                      corrosponding subgoal
 * Return a vector: cartesian product of views in each bucket
 *************************************************************/
        public static Vector cartPro(Vector subgoalVec, Vector[] bucketVecArr){

            Vector tempQuery1 = new Vector();

            int nOfPQuery = 1,    // number of potiential queries
            currentVecSize = 0,
            subNumber = 0; // number of subgoal
            String tempQuery2 = "", temp = "";
            subNumber = subgoalVec.size();

            StringTokenizer tk;
```

```
//get size of all potiential queries include duplicated
for (int i = 0; i < subNumber; i++)
{
	nOfPQuery *= bucketVecArr[i].size();
}

//fill the first temp query
for (int i = 0; i < bucketVecArr[0].size(); i ++)
{
	tempQuery1.addElement(bucketVecArr[0].elementAt(i));
}

//fill the potiential vector
for (int i = 1; i < subNumber; i++)
{
	currentVecSize = tempQuery1.size();

	for (int t = 0; t < currentVecSize; t++ )
	{
		tempQuery2 = tempQuery1.elementAt(0).toString() + " ";

		for ( int p = 0; p < bucketVecArr[i].size(); p++)
		{
			temp = tempQuery2;
			tempQuery2 += bucketVecArr[i].elementAt(p);
			tempQuery1.addElement(tempQuery2);
			tempQuery2 = temp;
		}
		tempQuery1.remove(0);
	}
}

Vector [] pQueryVec = new Vector[tempQuery1.size()];

// remove duplicated views in each vector
Vector tempVec = new Vector();
String tempStr1 = "", tempStr2 = "";

//int compareStr = 0;
for ( int i = 0; i < tempQuery1.size(); i ++ )
{
	tk = new StringTokenizer(tempQuery1.elementAt(i).toString());
	tempStr2 = tk.nextToken();
	tempVec.addElement(tempStr2);

	while(tk.hasMoreTokens())
	{
		tempStr1 = tk.nextToken();
```

```
                    if (checkBucket(tempStr1,tempVec))
                        tempVec.addElement(tempStr1);
                }

                pQueryVec[i] = tempVec;
                Vector rVec = new Vector();
                rVec = sortVecContent(tempVec);

                tempVec.removeAllElements();
                pQueryVec[i] = rVec;
                    //Got sorted non inside duplicate views,with duplicate whole views
            }

            Vector finalVec = new Vector(1);
            finalVec = reVector(pQueryVec);//remove duplicated potiential combinations

            return finalVec;

        }

/***********************************************
  * remove duplicate contents in vector.
  * Parameter @ ar includes duplicated contains
  ***********************************************/
        public static Vector reVector(Vector [] ar){

            int c = ar.length;
            String str = "", str2 = "";
            Vector vecTemp = new Vector(1);

            str = ar[0].toString();
            vecTemp.addElement(str);

            for (int i = 1; i < c; i++ ){
                str2 = ar[i].toString();
                if(checkBucket(str2, vecTemp)){
                    vecTemp.addElement(str2);
                }
            }
            return vecTemp;
        }

/***************************************************
  * set user query to query head and query body
  * Parameter @ quer is a whole user query
  ***************************************************/
        public static void setQuery(String quer){

            String head = "", tableName1 = "", var = "";
            StringTokenizer token;
```

```
        int h = 0,t = 0;
        h = quer.indexOf("FROM");
        t = quer.indexOf("WHERE");

        head = quer.substring(0, h).trim();//get whole head string
        head = head.substring(7);

        tableName1 = quer.substring(h, t).trim();//get table string
        tableName1 = tableName1.substring(5);

        token = new StringTokenizer(head, ",");//remove ","

        while (token.hasMoreTokens())
        {
            var = token.nextToken().trim(); // this is SELECT
            queryHead.addElement(var);
                    //add all variables to queryHead Vector.
        }

        token = new StringTokenizer(tableName1, ","); //remove ","

        while (token.hasMoreTokens())
        {
            var = token.nextToken().trim();
            queryTable.addElement(var);
        }
    }

/**************************************************
  * Check if Vector vec includes String str
  * Return true if it vec doesnot include str
  **************************************************/
    public static boolean checkBucket (String str, Vector vec){

        String temp = "";
        boolean flag = true;
        for (int i = 0; i < vec.size(); i ++ )
        {
            temp = vec.elementAt(i).toString();
            if (temp.equals(str)){
                flag = false;
                break;
            }
        }
        return flag;
    }

/**************************************************
  * sort the contents in one vector
  * parameter @ vec contains all unsort objects
```

```
 * Return a vector which contains sorted objects
 ************************************************/
    public static Vector sortVecContent(Vector vec){

        Vector vecM = new Vector(1);
        int i = vec.size();
        String [] str = new String[i];

        for (int t = 0; t < i; t ++ ){
            str[t] = vec.elementAt(t).toString();
        }

        Arrays.sort(str);
        vec.removeAllElements();
        for(int t = 0; t < i; t ++ ){
            vecM.addElement(str[t]);
        }
        return vecM;
    }

}// end class
```

VITA

Jing Hu

Candidate for the Degree of

Master of Science

Thesis: OPTIMIZING QUERIES USING A MATERIALIZED VIEW IN A DATA WAREHOUE

Major Field: Computer Science

Biographical:

Personal Data: Born in Fei Dong, Anhui, P.R.China, On March 20, 1976, the daughter of Xueming Hu and Damin Song.

Education: Graduated from Feidong No. 1 Middle School, Feidong, Anhui, P.R.China in July, 1993; Received Bachelor of Medicine degress in Clinical Medical from Anhui Medical University, Hefei, Anhui, P.R.China in July, 1998. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in July, 2006.

Experience: Employed as a business consultant in Ding Yuan Food Inc; Employed by Oklahoma State University , Department of Computer Science as a Teaching Assistant; Oklahoma State University, Department of Computer Science, 2002 to present.

Professional Memberships: N/A

Name: Jing Hu Date of Degree: July, 2006

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: OPTIMIZING QUERIES USING A MATERIALIZED VIEW IN A DATA WAREHOUE

Pages in Study: 53 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study: A data warehouse is a user-centered environment for data analysis and decision support. To support decision maker to make decision quickly and accurately, using materialized views can provide significant improvements in query processing time.

The problem of answering queries using views is to find efficient methods of answering a query using a set of previously materialized views over the database, rather than accessing the database relations. The known algorithms, the bucket algorithm, the inverse-rules algorithm have been used to rewrite queries using views before executing the queries. The bucket algorithm, predominantly used to rewrite queries, generates a candidate rewriting to a query using views and checks that the rewriting is contained in the original query. However, the bucket algorithm shows its deficiencies, we then describe the containment bucket algorithm and give optimal method to solve this problem. We present the experiment of comparing the performance of both algorithms.

Findings and Conclusions: This thesis presents a new algorithm for answering queries using materialized views. Materialized views can reduce query processing time very significantly but only of the query optimizer is able to find applicable views quickly. We began by analyzing the existing algorithm: bucket algorithm, and found that it has significant limitation. We developed the containment bucket algorithm, a better algorithm for answering queries using views and it outperforms the existing algorithm. The experiments showed that the bucket algorithm performed worse than containment bucket algorithm in both cases. In both cases the containment bucket algorithm outperformed the bucket algorithm by differing factors. The problem with the bucket algorithm is that discovers many of the interactions between the views in its second phase, and the performance in that phase is heavily dependent on the number of views in each query subgoal. Since performance depends heavily on the interaction with the views, the containment bucket algorithm gives optimal method to solve this problem. We developed the containment algorithm for answering queries using views and showed outperforms the bucket algorithms.

ADVISER'S APPROVAL: Dr. G. E. Hedrick